

A general variable neighborhood search for single-machine total tardiness scheduling problem with step-deteriorating jobs

Peng Guo^{a,*}, Wenming Cheng^a, Yi Wang^b

^a*School of Mechanical Engineering, Southwest Jiaotong University, Chengdu, China*

^b*Department of Mathematics, Auburn University at Montgomery, AL, USA*

Abstract

In this article, we study a single-machine scheduling problem of minimizing the total tardiness for a set of independent jobs. The processing time of a job is modeled as a step function of its starting time and a specific deteriorating date. A mixed integer programming model was applied to the problem and validated. Since the problem is known to be NP-hard, we proposed a heuristic named simple weighted search procedure (SWSP) and a general variable neighborhood search algorithm (GVNS). A perturbation procedure with 3-opt is embedded within the GVNS process in order to explore broader spaces. Extensive numerical experiments are carried out on some randomly generated test instances so as to investigate the performance of the proposed algorithms. By comparing to the results of the CPLEX optimization solver, the heuristic SWSP and the standard variable neighborhood search, it is shown that the proposed GVNS algorithm can provide better solutions within a reasonable running time.

Keywords: single-machine scheduling, general variable neighborhood search, step-deterioration, total tardiness.

1. Introduction

As an important tool for manufacturing and engineering, scheduling has a major impact on the productivity of a process. In the classical model of scheduling theory, it is assumed that the processing times of all jobs are known in advance and remain constant during the entire production process. However, this assumption may not be applicable to model some real manufacturing and service problems. Examples can be found in steel production, equipment maintenance, cleaning tasks allocation and so forth [21, 28], where any delay or waiting in starting to process a job may increase the necessary time for its completion. Such kinds of jobs are called *deteriorating jobs*. This kind of *scheduling problems with deteriorating jobs* was firstly considered by Browne and Yechiali [2] and Gupta and Gupta [8]. They assumed that the processing time of a job is a single linearly non-decreasing function of its starting time and aimed at minimizing the expected makespan and the variance of the makespan in a single-machine environment. The function that describes the processing time of a deteriorating job shall henceforth be called a *deterioration function*. Since then, there is a rapidly growing interest in the literature to study various types of scheduling problems involving deteriorating jobs. A recent survey of scheduling problems with deteriorating jobs was provided by Cheng et al. [5].

Most of the research concentrated on obtaining the optimal schedule for scheduling jobs for which their processing times continuously depend on their starting times [1, 31, 35, 17, 34, 13].

*Corresponding author. E-mail address: pengguo318@gmail.com

However, if some jobs fail to be processed prior to a pre-specified threshold, their processing times will be extended by adding an extra penalty time in some situations. Job processing times of this type of deterioration may be characterized by a piecewise defined function. Kunnathur and Gupta [21] firstly studied single-machine scheduling with a piecewise deterioration function. They assumed the actual processing time of a job is split into a fixed part and a variable penalty part. The variable part depends linearly upon the starting time of the job. Sundararaghavan and Kunnathur [33] introduced a total weighted completion time scheduling problem with step deterioration. Then, Mosheiov [29] investigated the scheduling problem with step deterioration in which the objective is to minimize the makespan on a single machine or multiple machines. He also introduced a simple heuristics for all these NP-hard problems.

For the scheduling problem with a piecewise linear deterioration function, Kubiak and Van de Velde [20] considered single-machine scheduling with a generalization of the unbounded deterioration model proposed by Browne and Yechiali [2] and presented NP-hardness proofs. They also developed a pseudo-polynomial dynamic programming algorithm and a branch and bound algorithm to obtain the optimal schedule. Alternatively, Cheng et al. [4] studied the problem of scheduling jobs with piecewise linear decreasing processing times on a single or multiple machines. They aimed to minimize the makespan and the *flow time*, i.e., the total completion time, and proved that the two problems are NP-hard. Subsequently, Ji and Cheng [16] gave a fully polynomial time approximation scheme for the same case with a single-machine. Afterwards, Wu et al. [36] provided two heuristic algorithms to solve the single-machine problem under the piecewise linear deterioration model. Moslehi and Jafari [30] dealt with the single-machine scheduling problem with the assumption of piecewise-linear deterioration. They suggested a branch and bound algorithm and a heuristic algorithm with complexity $O(n^2)$ for the objective of minimizing the total number of tardy jobs, where n is the total number of jobs in the problem.

Some articles focused on the scheduling problem with step-deterioration have been published. Cheng and Ding [3] studied some single-machine scheduling problems and showed that the *flow time problem* is NP-complete. Jeng and Lin [14] introduced a branch and bound algorithm for the single-machine problem of minimizing the makespan. In addition, They also studied the same problem for minimizing the flow time [15]. He et al. [11] proposed an exact algorithm to solve most of the problems with up to 24 jobs and a heuristic algorithm to derive a near-optimal solution. Layegh et al. [22] minimized the total weighted completion time by using a memetic algorithm. Cheng et al. [6] developed a variable neighborhood search to minimize the flow time on parallel machines. Furthermore, batch scheduling with step-deterioration has got attentions [24, 27]. However, *the total tardiness as one important objective in practice has not been concerned on the studies of single-machine scheduling problem with step-deterioration.*

Minimizing *total tardiness* has been drawing considerable attentions of researchers in the past decades and it is NP-hard in the strong sense [7]. The latest theoretical developments for the single-machine scheduling problem and the current state-of-the-art algorithms are reviewed by Koulamas [19]. He found that the single-machine total tardiness scheduling problem continues to attract researchers' interests from both theoretical and practical perspectives. *In this article, we consider the single-machine total tardiness scheduling problem with step-deteriorating jobs.* Since the total tardiness problem without step-deterioration in a single-machine is a NP-hard problem, the problem tackled here is also a NP-hard problem. To the best of our knowledge, there is no literature on minimizing the total tardiness in a single-machine scheduling problem with step-deteriorating jobs.

The remainder of the study is organized as follows. Section 2 contains the problem description and a mixed integer programming model. In Section 3, a heuristic and a general variable neighborhood search algorithm are developed for the proposed problem. The proposed methods are tested and compared to the CPLEX and the variable neighborhood search (VNS) on variance

test instances of both small size and large size in Section 4. In the last section, conclusions and future works are mentioned.

2. Problem formulation

The single-machine total tardiness scheduling problem with step deteriorating jobs can be described as follows. Let $N := \{1, 2, \dots, n\}$ be the set of n jobs to be scheduled on a single-machine without preemption. Assume that all jobs are ready at time zero and the machine is available at all times. In addition, the machine can handle only one job at a time, and cannot keep idle until the last job assigned to it is processed and finished. For each job $j \in N$, there is a *basic processing time* a_j , a *due date* d_j and a given *deteriorating threshold*, also called *deteriorating date* h_j . If the *starting time* s_j of job $j \in N$ is less than or equal to the given threshold h_j , then job j only requires a basic processing time a_j . Otherwise, an extra penalty b_j is incurred. Thus, the *actual processing time* p_j of job j can be defined as a step-function: $p_j = a_j$ if $s_j \leq h_j$; $p_j = a_j + b_j$, otherwise. Without loss of generality, the four parameters a_j , b_j , d_j and h_j are assumed to be integers.

Let $\pi = (\pi_1, \dots, \pi_n)$ be a sequence that arranges the current processing order of jobs in N , where π_k , $k = 1, \dots, n$, indicates the job in position k . The *tardiness* T_j of a job j in a schedule π can be calculated by $T_j = \max\{0, s_j + p_j - d_j\}$. The objective is to find a schedule π^* such that the *total tardiness* $\sum T_j$ is minimized. The total tardiness is a non-decreasing criterion of the job completion times. Using the three-field notation, this problem can be denoted by $1|p_j = a_j \text{ or } a_j + b_j, h_j|\sum T_j$.

Based on the above description, we formulate the problem as a 0-1 integer programming model. Firstly, the decision variable y_{ij} , $i, j \in N$ is defined such that y_{ij} is 1 if job i is scheduled before job j on the single-machine, and 0 otherwise. For each pair of jobs i and j , $y_{ij} + y_{ji} = 1$. Then, for a schedule π of the jobs in N , we minimize

$$Z(\pi) := \sum_{j=1}^n T_j \quad (2.1)$$

subject to

$$p_j = \begin{cases} a_j, & s_j \leq h_j \\ a_j + b_j, & \text{otherwise,} \end{cases} \quad \forall j \in N \quad (2.2)$$

$$s_i + p_i \leq s_j + M(1 - y_{ij}), \quad \forall i, j \in N, i \neq j \quad (2.3)$$

$$s_j + p_j - d_j \leq T_j, \quad \forall j \in N \quad (2.4)$$

$$y_{ij} \in \{0, 1\}, \quad \forall i, j \in N, i \neq j \quad (2.5)$$

$$s_j, T_j \geq 0, \quad \forall j \in N, \quad (2.6)$$

where M is a large positive constant such that $M \rightarrow \infty$ as $n \rightarrow \infty$. For example, M may be chosen as $M := \max_{j \in N} \{d_j\} + \sum_{j \in N} (a_j + b_j)$.

In the above mathematical model, equation (2.1) represents the objective of minimizing the total tardiness. Constraint (2.2) defines the processing time of each job. Constraint (2.3) determines the starting time s_j of job j with respect to the decision variables y_{ij} . Constraint (2.4) defines the tardiness of job j . Finally, constraints (2.5) and (2.6) define the boundary values of variables y_{ij} , s_j , T_j , for $i, j \in N$.

3. Solution methodologies

Due to the NP-hardness of the considered problem, only small size instances can be solved optimally by the enumeration techniques such as branch and bound or dynamic programming. However the size of some practical problem encountered in industry is usually large. We need to develop feasible heuristics to deal with large-sized instances. In this section, a simple weighted search procedure and a general variable neighborhood search algorithm are proposed.

Before introducing the algorithms, we discuss two properties of the considered problem that will be used later.

Property 3.1. *Consider any two jobs j and k that are processed before their given deteriorating dates. If $p_j \leq p_k$ and $d_j \leq d_k$, then job j must be scheduled before job k in an optimal sequence.*

Proof. Since the two jobs under consideration are not deteriorated, their processing times are only the basic processing times. Note that they remain to be not deteriorated even if the order of processing of the two jobs is switched. Lemma 3.4.1 [32, p. 51] regarding the total tardiness problem in a single-machine environment says if $p_j \leq p_k$ and $d_j \leq d_k$ then there exists an optimal sequence in which job j is scheduled before job k . Thus, applying this Lemma based on the basic processing times and the due dates yields the optimal sub-sequence including the job pair (j, k) in an optimal schedule. \square

Property 3.2. *Consider any two jobs j and k that are processed after their given deteriorating dates. If $p_j + b_j \leq p_k + b_k$ and $d_j \leq d_k$, then job j should be scheduled before job k in an optimal sequence.*

Proof. Since the two jobs j and k are processed after their deteriorating times, the actual processing time of each job is the sum of its basic processing time and a penalty time. Thus, like Property 3.1, Property 3.2 is a direct result of applying Lemma 3.4.1 of [32] to all such job pairs (j, k) based on their actual processing times $(a_j + b_j, a_k + b_k)$ and the due dates (d_j, d_k) . \square

According to the two properties, a job should be scheduled in an earlier position if it has a smaller value of a_i (or $a_i + b_i$ if it is deteriorated) and a smaller value of d_i . There is a weighted search algorithm that has applied to solve the single-machine flow time scheduling problem [11]. Thus, we adopt this idea and provide a simple weighted search procedure (SWSP) to obtain a heuristic for the $1|p_j = a_j \text{ or } a_j + b_j, h_j| \sum T_j$ problem.

3.1. Simple weighted search procedure

We shall need an appropriately chosen triple $\omega := (\omega_1, \omega_2, \omega_3)$ of positive weights to compute a weighted value $m_i := \omega_1 d_i + \omega_2 p_i + \omega_3 h_i$, for each $i \in N$. Since it is difficult to determine a priori a triple of weights that can yield good solutions, we adopt the *dynamic update strategy* to search all possible triples ω . Weight ω_1 is a linearly increasing weight, given by $\omega_1 = \omega_{1\min} + (\omega_{1\max} - \omega_{1\min})(l_1 - 1)/(n - 1)$, where l_1 varies from 1 to n . Weight ω_2 adopts the same updating formula as weight ω_1 . The only difference between the two weights is their possible minimal and maximal values. Suitable values of $\omega_{1\min}$, $\omega_{1\max}$, $\omega_{2\min}$ and $\omega_{2\max}$ may be determined by preliminary experiments. For example, for the numerical experiments we conducted in this article, we have chosen the values of the four parameters $\omega_{1\min}$, $\omega_{1\max}$, $\omega_{2\min}$ and $\omega_{2\max}$ to be 0.2, 0.9, 0.1 and 0.7, respectively. With ω_1 and ω_2 determined, $\omega_3 := 1 - \omega_1 - \omega_2$. But ω_3 may become negative by using this formula. Should a negative ω_3 occurs, it is adjusted to a pre-specified positive value. In this article, this preset value is chosen to be 0.1. Let Ω be the set of all possible triples of weights obtained.

We now describe the SWSP. Firstly, let a triple $(\omega_1, \omega_2, \omega_3) \in \Omega$ be fixed. Then the job with the minimal due date is scheduled in the first position. Subsequently the unscheduled jobs are arranged in a nondecreasing order of the weighted sums m_i , $i \in N$. Thus each triple $(\omega_1, \omega_2, \omega_3) \in \Omega$ is associated with a schedule. Correspondingly, for each schedule, the value of the objective total tardiness is calculated. Among all these different solutions, the best solution is given by the one with the smallest total tardiness. Furthermore, in order to refine the quality of the solution, a local improvement approach based on pairwise swapping of jobs is used in the procedure. The detailed SWSP is described in Algorithm 1.

Algorithm 1 Simple weighted search procedure (SWSP)

```

1: Input the initial data of a given instance;
2: Set  $c_{[0]} = 0$ ,  $N_0 = \{1, \dots, n\}$  and  $\pi = []$ ;
3: Generate the entire set  $\Omega$  of possible triples of weights. For each triple  $(\omega_1, \omega_2, \omega_3) \in \Omega$ ,
   perform the following steps;
4: Choose job  $i$  with minimal due date to be scheduled in the first position;
5:  $c_{[1]} = c_{[0]} + a_i$ ,  $\pi = [\pi, i]$ ;
6: delete job  $i$  from  $N_0$ ;
7: set  $k = 2$ ;
8: repeat
9:   if  $c_{[k-1]} > \max\{h_r, r \in N_0\}$  then
10:    choose job  $j$  from  $N_0$  with the smallest weighted value  $m_j$  to be scheduled in the  $k$ th
    position;
11:     $c_{[k]} = c_{[k-1]} + a_j + b_j$ ,  $\pi = [\pi, j]$ ;
12:    delete job  $j$  from  $N_0$ ;
13:     $k = k + 1$ ;
14:  else
15:    choose job  $j$  from  $N_0$  with the smallest value  $m_j$  to be scheduled in the  $k$ th position;
16:    if  $c_{[k-1]} > h_j$  then
17:       $c_{[k]} = c_{[k-1]} + a_j + b_j$ ;
18:    else
19:       $c_{[k]} = c_{[k-1]} + a_j$ ;
20:    end if
21:     $\pi = [\pi, j]$ ;
22:    delete job  $j$  from  $N_0$ ;
23:  end if
24: until the set  $N_0$  is empty
25: calculate the total tardiness  $Z(\pi)$  of the obtained schedule  $\pi$ ;
26: for  $i \leftarrow 1$  to  $n$  do
27:   for  $j \leftarrow 1$  to  $n$  do
28:    if  $i \neq j$  then
29:      create a new solution  $\pi'$  by interchanging jobs in the  $i$ th and  $j$ th position from  $\pi$ ;
30:      replace  $\pi$  by  $\pi'$  if the total tardiness of  $\pi'$  is smaller than that of  $\pi$ ;
31:    end if
32:   end for
33: end for
34: Output the final solution  $\pi$ .

```

3.2. General variable neighborhood search heuristic

The variable neighborhood search (VNS) is a simple and effective meta-heuristic proposed by Mladenović and Hansen [25]. It has the advantage of avoiding entrapment at a local optimum by systematically swapping neighborhood structures during the search. The basic VNS combines two search approaches: a stochastic approach in the *shaking step* that finds a random neighbor of the incumbent, and a deterministic approach which applies any type of local search algorithm. Several VNS variants have been proposed and successively applied to numerous combinatorial optimization problems [10]. Among which, the variable neighborhood descent (VND) [9] as the best improved deterministic method is the most frequently used variant. The VND initiates from a feasible solution as the incumbent one and then carries out a series of neighborhood searches through operators $\mathcal{N}_k, (k = 1, \dots, k_{\max})$. If a better solution is found in a neighborhood, the incumbent solution is replaced by the better one and the search continues within the current neighborhood; otherwise it will explore the next neighborhood. The obtained solution at the end of the search is a local optimum with respect to all neighborhood structures. If the deterministic local search approach of the basic VNS is replaced by the VND search, the resulted algorithm is called a *general VNS* (GVNS). The GVNS has received attentions and showed good performance compared to other VNS variants. So far, the GVNS has been applied to many optimization problems, such as the incapacitated single allocation p -hub median problem [12], the one-commodity pickup-and-delivery traveling salesman problem [26], the capacitated vehicle routing problem [23] and the single-machine total weighted tardiness problem with sequence dependent setup times [18].

As far as we know, there is no published work on solving scheduling problems with deteriorating jobs by the GVNS. Therefore, in this section *we develop a GVNS heuristic for the single-machine total tardiness scheduling problem with step-deterioration*. In what follows, the main components of the designed GVNS algorithm are described: the initialization process, five neighborhood structures provided for the shaking step and local search procedures. In addition, a perturbation phase with a 3-opt operation without change of direction is embedded within the search process in order to decrease the probability of returning to the previous local optimum.

3.2.1. Initialization

Typically, a good initial sequence can be obtained by using the Earliest Due Date (EDD) rule. That is to say, all jobs are arranged in nondecreasing order of the due dates.

3.2.2. Neighborhood structures

In a local search algorithm, a neighborhood structure is designed by introducing moves from one solution to another. In order to conduct a local search in the proposed GVNS, we next develop five neighborhood structures based on swap, insertion and k -opt for the problem $1|p_j = a_j \text{ or } a_j + b_j, h_j| \sum T_j$. These neighborhood structures are defined by their corresponding operators. We remark that for simplicity a neighborhood structure shall be referred to by the name of its corresponding generating operator. For example, if a neighborhood structure is generated by an operator \mathcal{N}_1 , the neighborhood structure shall be called \mathcal{N}_1 neighborhood structure or simply \mathcal{N}_1 neighborhood.

Swap Operator (\mathcal{N}_1): The swap operator (Figure 3.1) selects a pair of jobs π_i and π_j in the current sequence π of jobs, exchanges their positions, and computes the total tardiness of the resulting solution. If an improvement is found, the new sequence is accepted as the incumbent one. The operator repeats this process for all indices $i, j \in N$, with $i \neq j$ until all the neighborhoods have been searched. The size of the swap operator is $n(n-1)/2$.

Insertion Operator (\mathcal{N}_2): For a given incumbent arrangement of jobs, the insertion neighborhood (Figure 3.2) can be obtained by removing a job from its position and inserting it into

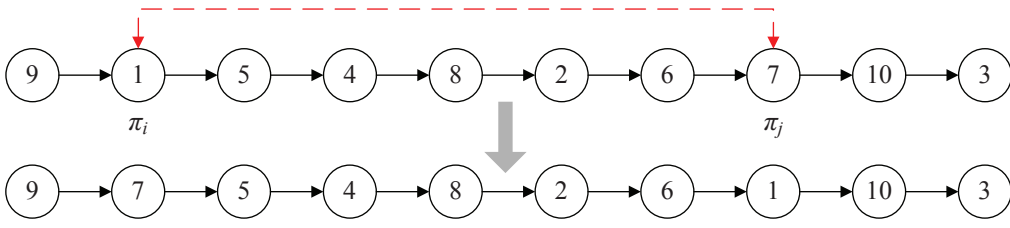


Figure 3.1: Illustration of the swap operator

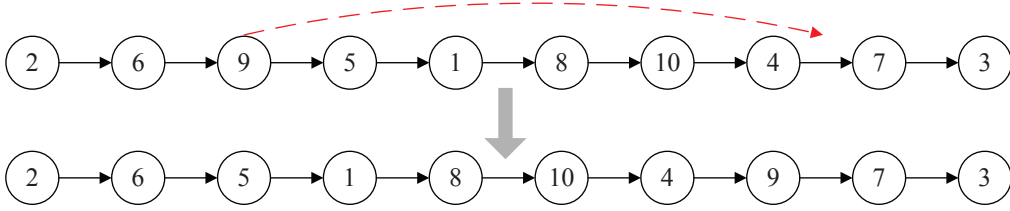


Figure 3.2: Illustration of the insertion operator

another position. All jobs are considered for insertion operation. Once an improvement is observed, the new arrangement of jobs is deemed as the incumbent one.

Pairwise Exchange Operator (\mathcal{N}_3): The pairwise exchange operator is similar to the swap operator except that the swap is applied to a pair of jobs rather than a single job. In our implementation, these pairwise jobs are selected from all the combinations of two out of n jobs. It exchanges their positions and computes the total tardiness of the resulting sequence. Once an improvement is obtained, the incumbent solution is updated by the new solution.

Couple Insertion Operator (\mathcal{N}_4): This procedure is similar to the insertion operator \mathcal{N}_2 . But the operation is for a pair of successive jobs. For each couple of jobs π_i and π_{i+1} , $1 \leq i < n - 1$, the operator extracts these two jobs and inserts them in another pair of positions j and $j + 1$, $1 \leq j \leq n - 1$. Note that $i \neq j$.

2-opt Operator (\mathcal{N}_5): The 2-opt is the most classical heuristic for the traveling salesman problem in which it removes two edges from the tour and reconnects the two paths created. In our implementation (Algorithm 2), the 2-opt operator selects two jobs π_i and π_j in the current sequence. It then deletes the edge connecting job π_i and its successor and the edge connecting job π_j with its successor. Afterwards, it constructs a new connection of π_i to π_j and a new connection between their respective successors. Furthermore, the partial sequence between the successors of π_i and π_j is reversed. Figure 3.3 illustrates an example of the 2-opt procedure. When an improvement is found in terms of the objective value, the incumbent is updated with the improved sequence. The search continues with applying the 2-opt operator to all possible pairs of jobs that are at least *three* positions away from each other. The solution obtained by this operator may not significantly differ from the incumbent in terms of the objective values. But some random jumps in the objective value may be achieved to escape from a current local optimum due to the path reversion.

3.2.3. Shaking and local search

A shaking operation is performed before the local search in each iteration. The shaking procedure plays a role in diversifying the search and facilitating escape from a local optimum. To preserve the simplicity of the principles of the VNS, both the shaking procedure and the local search of the GVNS make use of the same set \mathcal{N}_k ($k=1, \dots, 5$) of neighborhood structures. The

Algorithm 2 2-opt Operator (\mathcal{N}_5)

```
1: for  $i \leftarrow 1$  to  $n$  do
2:   for  $j \leftarrow 1$  to  $n$  do
3:      $I \leftarrow \min(i, j)$ ;
4:      $J \leftarrow \max(i, j)$ ;
5:     if  $J - I \geq 3$  then
6:       apply the 2-opt procedure to the jobs  $\pi_i$  and  $\pi_j$  to generate a new sequence  $\pi'$ ;
7:       if the new sequence  $\pi'$  is better than the current sequence in terms of objective value
         then
8:         update the incumbent;
9:       end if
10:    end if
11:  end for
12: end for
```

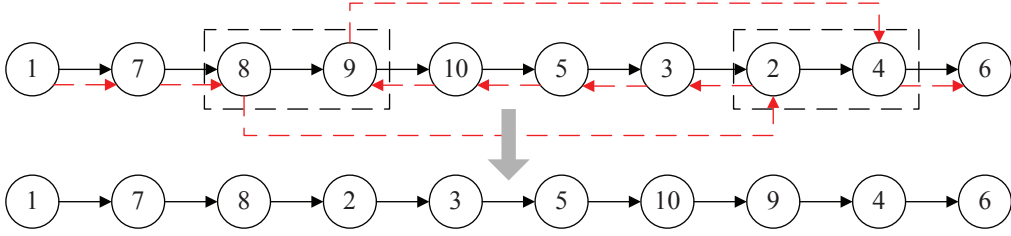


Figure 3.3: Illustration of the 2-opt operator (\mathcal{N}_5)

shaking operation is implemented by generating randomly a neighboring solution π' of the current one π using a given neighborhood operator \mathcal{N}_k . The neighborhood operator \mathcal{N}_k will be chosen to cycle from \mathcal{N}_1 through \mathcal{N}_5 . If the given neighborhood structure is \mathcal{N}_1 or \mathcal{N}_5 , then the shaking procedure randomly selects two jobs. If the given neighborhood structure is \mathcal{N}_2 , then the shaking procedure randomly selects a job and an insertion position. If the given neighborhood structure is \mathcal{N}_3 , then the shaking procedure randomly selects two pairs of jobs. If the given neighborhood structure is \mathcal{N}_4 , then the shaking procedure randomly selects a couple of two consecutive jobs and a couple of two consecutive positions.

A complete local search is organized as a VND using the proposed neighborhood structures. To efficiently explore possible solutions, a sequential order K of applying these neighborhoods are randomly generated. For instance, assuming that the sequence is $K = (3, 1, 2, 5, 4)$, the search starts from \mathcal{N}_3 and ends at \mathcal{N}_4 . For each neighborhood, a new local optimum π'' is obtained by carrying out the corresponding local search operation. If π'' is better than π' , the new solution π'' will be accepted as a descent so that π' is updated with π'' , and the search continues for a new π'' within the current neighborhood; otherwise the search turns to the next neighborhood in K . The search stops until the last neighborhood in K is explored. It is worthwhile to point out that after one iteration of a shaking and local search, the solution π' generated by the shaking procedure may not be improved.

The shaking and local search steps are summarized below.

1. Begin the shaking procedure. Use the current solution π to randomly generate a neighbor π' and set $i = 1$.
2. Obtain the sequential order K of neighborhood search at random.

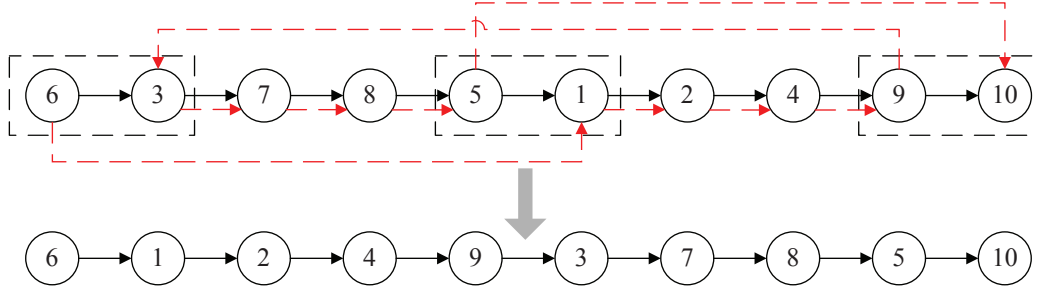


Figure 3.4: Illustration of the 3-opt moves (\mathcal{N}_5)

3. Apply the neighborhood operator $\mathcal{N}_{K(i)}$ to π' to achieve a local optimum π'' , where $K(i)$ means the i -th entry of the sequence K .
4. If π'' is better than π' , update π' and keep the value i so the search will continue within the current neighborhood; otherwise, $i \leftarrow (i + 1)$.
5. If $i > k_{\max} = 5$, complete one iteration of the shaking and local search; otherwise, go to Step 3.

3.2.4. Perturbation phase

Since a local search applied to optimization problems often suffers from getting trapped in a local optimum, the well-know approach for this deficiency is to adopt a multi-start method when no improvement is observed. Multi-start heuristics can usually be characterized as iterative procedures consisting of two phases: the first phase in which a starting solution is generated and the second one in which the starting solution is typically (but not necessarily) improved by local search methods. However, a far more effective approach is to generate the new starting solution from an obtained local optimum by a suitable perturbation method.

Inspired by the multi-start strategy, we provide a perturbation phase with 3-opt operator *without change of direction* for producing a new starting solution in case a local search is trapped in a local optimum. In our implementation, if the current solution π cannot be further improved through a predetermined number γ of iterations of the shaking and local search procedures, the GVNS algorithm assumes that there is no hope to continue the local search based on the current best solution π . Then, the 3-opt operator (Figure 3.4) is implemented on π . Three jobs are randomly selected from the incumbent sequence. The 3-opt operator removes the three edges connecting the selected jobs with their successors. Then it reconnects the four subsequences created. The direction of these subsequences remains unchanged because reversing the direction of these subsequences may produce a much worse solution after a number of local searches.

3.2.5. Proposed GVNS algorithm

The GVNS algorithm starts with an initialization phase in which an initial solution π_0 is generated by the EDD rule and is set as the current solution π . To evaluate the quality of solutions, the solution value is considered as the total tardiness.

At each iteration, the proposed algorithm needs to carry out mainly two steps. First, a shaking procedure is performed to generate a random neighboring solution π' of the incumbent solution π . Subsequently, a local search based on the VND with π' as the input solution is performed in an attempt to improve π' . The local solution π' , possibly improved by the local search is then compared to the current best solution π in terms of the objective function value. If π' is better than π , the current solution π is updated by π' . This completes one iteration of

the shaking and local search. The GVNS algorithm then continues with next iteration of shaking and local search. In addition, if no improvement of the current best solution π can be found through a predetermined number γ of iterations of shaking and local search, a 3-opt perturbation procedure is used to generate a newly starting solution π .

We next describe the *stopping criterion*. A pre-specified maximum number $Iter_{\max}$ of iterations of shaking and local search or a maximum number $Iter_{\text{nip}}$ between two iterations without improvement are used as the stopping criterion of the algorithm. In our implementation, we choose $Iter_{\max} = 500$ and $Iter_{\text{nip}} = 150$, respectively. Moreover, the predetermined number γ of iterations to perform a 3-opt perturbation is chosen to be $\gamma = 0.5 \cdot Iter_{\text{nip}}$. This strategy is determined by our preliminary experiments. A Pseudo-code of the proposed GVNS algorithm is summarized in Algorithm 3.

Algorithm 3 General variable neighborhood search heuristic

- 1: Initialize the parameters of the algorithm;
 - 2: Define a set of neighborhood structures $\mathcal{N}_k(k = 1, \dots, 5)$, that will be used in the shaking phase and the local search phase;
 - 3: Generate the initial solution π_0 by the EDD rule;
 - 4: Calculate the objective value $f(\pi_0)$ for the solution π_0 ;
 - 5: Set the current best solution $\pi = \pi_0$;
 - 6: Choose the stopping criterion; initialize the counter: $iter_1 = 0$, $iter_2 = 0$ and $iter_3 = 0$;
 - 7: Set the first neighborhood structure for the shaking procedure to be $k \leftarrow 1$;
 - 8: **repeat**
 - 9: **Shaking:** Generate a point π' at random in the \mathcal{N}_k neighborhood of π ;
 - 10: Produce a random sequence K of applying the 5 neighborhood structures;
 - 11: **Local search:** Apply the VND scheme in the order specified by K , update π' if a better local optimum is obtained;
 - 12: **if** $f(\pi') < f(\pi)$ **then**
 - 13: $\pi \leftarrow \pi'$
 - 14: reset the counter $iter_2 = 0$;
 - 15: reset the counter $iter_3 = 0$;
 - 16: **else**
 - 17: Increment the counters: $iter_2 = iter_2 + 1$, $iter_3 = iter_3 + 1$;
 - 18: **end if**
 - 19: Update the counter of total number of iterations $iter_1 = iter_1 + 1$;
 - 20: **if** π has not improved for a given number of iterations γ , that is, when $iter_3 > \gamma$, and if $iter_2 < iter_{\text{NIP}}$ **then**
 - 21: use the 3-opt perturbation procedure to generate a new starting solution π ; and reset $iter_3 = 0$;
 - 22: **end if**
 - 23: Set $k = k \bmod 5 + 1$ to cycle through the neighborhood structures for shaking.
 - 24: **until** the stopping criterion is met, that is, $iter_1 > Iter_{\max}$ or $iter_2 > Iter_{\text{nip}}$;
 - 25: Output the current best solution π .
-

4. Computational experiments

In this section, in order to evaluate the performance of the GVNS algorithm and the proposed heuristic SWSP, a set of testing instances were generated and solved on a PC with Intel Core i3

3.20 GHz CPU and 4 GB of memory in the environment of Windows 7 OS. The procedure of generating the testing instances and analysis of the results are described below.

4.1. Experiments design

In this article, the basic processing times (a_j) are randomly generated from a discrete uniform distribution on the interval $(0, 100]$. The deteriorating dates (h_j) are picked from the uniform distributions over three different intervals $(0, A/2]$, $[A/2, A]$ and $(0, A]$, where, the value of A is obtained from the equation $A = \sum_{j \in N} a_j$. The deterioration penalties $b_j, j \in N$, are drawn from the uniform distribution $(0, 100 \times \tau]$, where we choose $\tau=0.5$. In addition, the due dates are randomly generated according to a discrete uniform distribution from two different intervals $(0, 0.5 \times \bar{C}_{\max}]$, and $(0, \bar{C}_{\max}]$, where \bar{C}_{\max} is the value of the makespan obtained by arranging the jobs in the non-decreasing order of the ratios $a_j/b_j, j \in N$ [1].

The algorithms were tested over five different job sizes for small-sized instances. Those sizes are respectively $n= 8, 10, 15, 20$ and 25 . For all possible combinations of deteriorating dates and due dates, to account for the three intervals in which deteriorating dates are generated and the two intervals in which due dates are drawn, six types of problems are needed to solve for a specific job size. For convenience, these types of problems are denoted by symbols $S_{k_1 k_2}$. For example, S_{12} represents a type of problems with deteriorating dates drawn from the interval $(0, A/2]$ and due dates drawn from the interval $(0, \bar{C}_{\max}]$. In addition, a set of large sized instances with $n=\{50, 60, 70, 80, 90, 100\}$ are being considered. As a consequence, for both small-sized and large-sized problems, $66 ((5 + 6) \times 6)$ sample instances were randomly generated.

4.2. Experimental results

Because there are no comparative data and no competing heuristic for our problem, comparisons with best know solutions are not possible. Thus, we have designed and coded the standard VNS heuristic (Algorithm 4) as a comparison to assess our approach. The VNS heuristic does not include the perturbation phase. The implementation sequence of neighborhoods is chosen by the deterministic order $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4 \text{ and } \mathcal{N}_5)$. The stopping criteria are chosen to be identical to the GVNS algorithm.

The described problem can be solved optimally by a commercial solver CPLEX 12.5 for some small-sized instances. However, because of its NP-hardness, it is impossible to obtain optimal solution by the CPLEX for medium or large instances. The preset run time for CPLEX is one hour. If CPLEX fails to either converge to the optimum or prove the optimality of the incumbent within the time limit, a GAP is provided by the software. The GAP shows the quality of a solution given by the CPLEX within the run time limit to some extent. A large GAP implies that the CPLEX needs more time to converge to an optimal solution.

All proposed algorithms are implemented in Matlab 7.11. The CPLEX and the SWSP are deterministic hence only one run is necessary for each problem instance. While the VNS and the GVNS are stochastic so we have to run some replications in order to better assess the experimental results. In our experiment, for each instance we run 10 times when using the VNS and the GVNS.

Each algorithm's performance is measured by computing a relative percentage deviation (RPD) defined by the equation

$$RPD(\%) = \frac{Z_{\text{alg}} - Z_{\text{best}}}{Z_{\text{best}}} \times 100 \quad (4.1)$$

where Z_{alg} is the solution value obtained for a given algorithm and instance, Z_{best} is the best solution of all approaches. For the VNS and the GVNS, the comparison results are in terms of average RPD and average computational time.

Algorithm 4 Variable neighborhood search heuristic

```
1: Initialize the parameters of the algorithm;
2: Define a set of neighborhood structures  $\mathcal{N}_k, k = 1, \dots, 5$ ;
3: Generate an initial solution  $\pi_0$  by the EDD rule;
4: Set the current best solution  $\pi = \pi_0$ ; initialize the counters:  $iter_1 = 0, iter_2 = 0$ .
5:  $k \leftarrow 1$ ;
6: repeat
7:   Shaking: Generate randomly a point  $\pi'$  in the  $k$ th neighborhood of  $\pi$ ;
8:   Local search: Apply neighborhood search operator  $\mathcal{N}_k$  with  $\pi'$  as initial solution to obtain
      a local optimum  $\pi''$ ;
9:   if  $f(\pi'') < f(\pi)$  then
10:     $\pi \leftarrow \pi''$ ;
11:    continue the local search with the current neighborhood  $\mathcal{N}_k$ ;
12:    reset the counter  $iter_2 = 0$ ;
13:   else
14:     $k \leftarrow k \bmod 5 + 1$ ;
15:    Increment the counter:  $iter_2 = iter_2 + 1$ ;
16:   end if
17:   Update the counter of total number of iterations  $iter_1 = iter_1 + 1$ ;
18: until the stopping criterion is reached, that is,  $iter_1 > Iter_{\max}$  or  $iter_2 > Iter_{\text{nip}}$ ;
19: Output the current best solution  $\pi$ .
```

Table 1 summaries the results obtained from the CPLEX, the SWSP, the VNS and the GVNS. As shown in the table, the GVNS finds the best solutions for all small-sized instances. Since the computational time of the CPLEX was preset to one hour, the CPLEX could not find the optimal solutions for all instances with 20 or 25 jobs. The computational time of the CPLEX exponentially increases as the number of jobs increases due to the NP-hardness of the problem. Note that 4 solutions obtained by the CPLEX are worse than those obtained by the GVNS. The average RPD given by the SWSP is 11.63%. The run times of the SWSP are very short and are ignored in table 1. Compared to the CPLEX, the run times of the GVNS and the VNS do not significantly increase as the number of jobs increases. The computational time of the GVNS is longer than that of the VNS because the GVNS includes the VND scheme as the local search and additionally the perturbation phase. However, the VNS delivers the best solutions for only 13 out of the 30 small-sized test instances. The RPD delivered by VNS is only 2.32%. This suggests that these designed neighborhood structures are well suitable for solving the problem under consideration.

For large-sized instances, it is impossible to find the optimal solution by using the CPLEX within the preset one-hour time limit. Thus we tested the performance of the proposed algorithms with respect to the standard VNS. The results are shown in Table 2. The average *RPD* of the GVNS is about 0.78% which is significantly smaller than that of the VNS. Except for instance S_{12} with 60 jobs, the RPDs of other instances given by the GVNS are very small. The relative higher RPD of instance S_{12} may be because its optimal solution value is relatively small and there are 2 local optima found in 10 replications. According to figure 4.1, it is observed that *the quality of solutions delivered by the GVNS is very good without respect to the distribution of the deteriorating dates*. On the other hand, the quality of solutions obtained by the SWSP is inferior compared to the GVNS and the VNS.

To measure the robustness of the algorithm, a mean absolute deviation (MAD) of 10 runs when applying the VNS or the GVNS was calculated for each of large-sized instances according

to the equation

$$MAD(\%) = \frac{1}{R \times \bar{Z}_{\text{alg}}} \sum_{r=1}^R (Z_{\text{alg}}(r) - \bar{Z}_{\text{alg}}) \times 100 \quad (4.2)$$

where R is the number of replications, \bar{Z}_{alg} denotes the average solution value obtained from the R runs for a given algorithm, and $Z_{\text{alg}}(r)$ indicates the solution value obtained for a given algorithm in the r -th run. The average MAD of the GVNS is merely 0.81%. Figure 4.2 reports the MAD obtained by the GVNS for large-sized instances. Overall, the GVNS demonstrates to be a good alternative to solve single-machine total tardiness scheduling problems with step-deteriorating jobs, because the algorithm finds good solutions in a reasonable computational time.

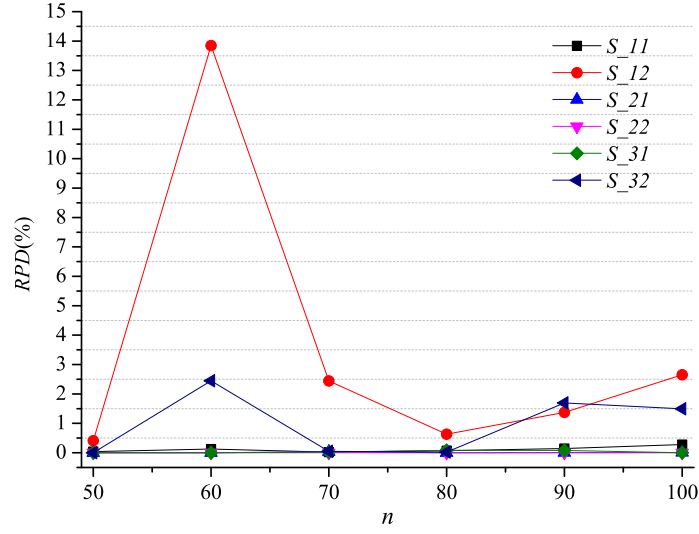


Figure 4.1: The RPDs given by the GVNS for large-sized instances

5. Conclusions

In this article, we considered a single-machine total tardiness scheduling problem with step-deteriorating jobs. The objective of this problem is to determine the sequence policy of the jobs under consideration so as to minimize the total tardiness. In order to solve the problem, an integer programming model was developed. Solutions were obtained by the CPLEX up to the size of 25 jobs. Due to the intractability of the problem, it is impossible to solve large instances to optimality by using the CPLEX. Based on two properties introduced, a heuristic called the SWSP and the GVNS algorithm were proposed to obtain near-optimal solutions of the problem. Meanwhile, the standard VNS algorithm was presented as reference for evaluating the performance of our algorithms. Experimental results showed that the proposed GVNS outperforms other heuristics in terms of relative percentage deviation from the best solution for both small- and large-sized instances. The computational times are in general short for the GVNS and the VNS. The GVNS performed better than the VNS in both cases. Furthermore, the GVNS is more robust than the VNS with regards to the choice of different intervals of deteriorating dates.

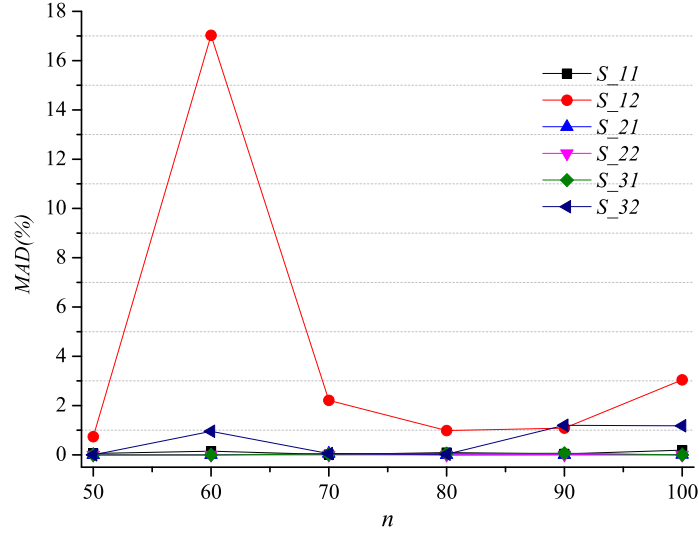


Figure 4.2: MAD given by the GVNS for large-sized instances

For further study, a suggestion is to consider the setup times between different jobs for the single-machine scheduling problem with step-deteriorating jobs. In addition, multi-machine or multi-criteria cases with step-deterioration are also encouraged.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (No.51175442) and the Fundamental Research Funds for the Central Universities (No. 2010ZT03; SWJTU09CX022).

References

References

- [1] A. Bachman and A. Janiak. Minimizing maximum lateness under linear deterioration. *European Journal of Operational Research*, 126(3):557–566, 2000.
- [2] S. Browne and U. Yechiali. Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498, 1990.
- [3] T.C.E. Cheng and Q. Ding. Single machine scheduling with step-deteriorating processing times. *European Journal of Operational Research*, 134(3):623–630, 2001.
- [4] T.C.E. Cheng, Q. Ding, M.Y. Kovalyov, A. Bachman, and A. Janiak. Scheduling jobs with piecewise linear decreasing processing times. *Naval Research Logistics*, 50(6):531–554, 2003.
- [5] T.C.E. Cheng, Q. Ding, and B.M.T. Lin. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1):1–13, 2004.

- [6] W. Cheng, P. Guo, Z. Zhang, M. Zeng, and J. Liang. Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs. *Mathematical Problems in Engineering*, 2012:1–20, 2012. doi: 10.1155/2012/928312.
- [7] J. Du and J.Y.T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.
- [8] J.N.D. Gupta and S.K. Gupta. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering*, 14(4):387–393, 1988.
- [9] P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.
- [10] P. Hansen, N. Mladenović, and J.A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.
- [11] C.C. He, C.C. Wu, and W.C. Lee. Branch-and-bound and weight-combination search algorithms for the total completion time problem with step-deteriorating jobs. *Journal of the Operational Research Society*, 60(12):1759–1766, 2009.
- [12] A. Ilić, D. Urošević, J. Brimberg, and N. Mladenović. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 206(2):289–300, 2010.
- [13] A. Jafari and G. Moslehi. Scheduling linear deteriorating jobs to minimize the number of tardy jobs. *Journal of Global Optimization*, 54(2):389–404, 2012.
- [14] A.A.K. Jeng and B.M.T. Lin. Makespan minimization in single-machine scheduling with step-deterioration of processing times. *Journal of the Operational Research Society*, 55(3):247–256, 2004.
- [15] A.A.K. Jeng and B.M.T. Lin. Minimizing the total completion time in single-machine scheduling with step-deteriorating jobs. *Computers and Operations Research*, 32(3):521–536, 2005.
- [16] M. Ji and T.C.E. Cheng. An fptas for scheduling jobs with piecewise linear decreasing processing times to minimize makespan. *Information Processing Letters*, 102(2-3):41–47, 2007.
- [17] M. Ji and T.C.E. Cheng. Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science*, 410(38-40):3761–3768, 2009.
- [18] G. Kirlik and C. Oguz. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers and Operations Research*, 39(7):1506–1520, 2012.
- [19] C. Koulamas. The single-machine total tardiness scheduling problem: Review and extensions. *European Journal of Operational Research*, 202(1):1–7, 2010.
- [20] W. Kubiak and S. Van de Velde. Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics*, 45(5):511–523, 1998.
- [21] A.S. Kunnathur and S.K. Gupta. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47(1):56–64, 1990.

- [22] J. Layegh, F. Jolai, and M.S. Amalnik. A memetic algorithm for minimizing the total weighted completion time on a single machine under step-deterioration. *Advances in Engineering Software*, 40(10):1074–1077, 2009.
- [23] H. Lei, G. Laporte, and B. Guo. A generalized variable neighborhood search heuristic for the capacitated vehicle routing problem with stochastic service times. *Top*, 20(1):99–118, 2012.
- [24] J. Y.T. Leung, C.T. Ng, and T.C.E. Cheng. Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times. *European Journal of Operational Research*, 187(3):1090–1099, 2008.
- [25] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
- [26] N. Mladenović, D. Urošević, S. Hanafi, and A. Ilić. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1):270–285, 2012.
- [27] B. Mor and G. Mosheiov. Batch scheduling with step-deteriorating processing times to minimize flowtime. *Naval Research Logistics*, 59(8):587–600, 2012.
- [28] G. Mosheiov. V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39(6):979–991, 1991.
- [29] G. Mosheiov. Scheduling jobs with step-deterioration; minimizing makespan on a single- and multi-machine. *Computers and Industrial Engineering*, 28(4):869–879, 1995.
- [30] G. Moslehi and A. Jafari. Minimizing the number of tardy jobs under piecewise-linear deterioration. *Computers and Industrial Engineering*, 59(4):573–584, 2010.
- [31] D. Oron. Single machine scheduling with simple linear deterioration to minimize total absolute deviation of completion times. *Computers and Operations Research*, 35(6):2071–2078, 2008.
- [32] M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, fourth edition, 2012.
- [33] P.S. Sundararaghavan and A.S. Kunnathur. Single machine scheduling with start time dependent processing times: Some solvable cases. *European Journal of Operational Research*, 78(3):394–403, 1994.
- [34] D. Wang and J.B. Wang. Single-machine scheduling with simple linear deterioration to minimize earliness penalties. *International Journal of Advanced Manufacturing Technology*, 46(1-4):285–290, 2010.
- [35] C.C. Wu, W.C. Lee, and Y.R. Shiau. Minimizing the total weighted completion time on a single machine under linear deterioration. *International Journal of Advanced Manufacturing Technology*, 33(11-12):1237–1243, 2007.
- [36] C.C. Wu, Y.R. Shiau, L.H. Lee, and W.C. Lee. Scheduling deteriorating jobs to minimize the makespan on a single machine. *International Journal of Advanced Manufacturing Technology*, 44(11-12):1230–1236, 2009.

Table 1: Comparison of the CPLEX, the SWSP, the VNS and the GVNS for small-sized instances.

Groups	n	CPLEX			SWSP		VNS			GVNS		
		Objective Value	Time(s)	GAP(%)	Objective Value	RPD(%)	Mean	RPD(%)	Time(s)	Mean	RPD(%)	Time(s)
$S_{_11}$	8	585	0.12	0.00	585	0.00	585	0.00	0.08	585	0.00	0.75
	10	638	0.87	0.00	680	6.58	645	1.10	0.12	638	0.00	1.44
	15	1891	1621.10	0.00	2129	12.59	1896.2	0.27	0.37	1891	0.00	3.75
	20	3062	3600.00	60.79	3427	11.92	3095.8	1.10	0.74	3062	0.00	7.70
	25	5476	3600.00	79.40	6048	16.17	5236.4	0.58	1.49	5206	0.00	16.63
$S_{_12}$	8	301	0.08	0.00	301	0.00	301	0.00	0.07	301	0.00	0.68
	10	570	0.30	0.00	595	4.39	590.8	3.65	0.12	570	0.00	1.40
	15	739	57.00	0.00	772	4.47	739	0.00	0.25	739	0.00	3.24
	20	343	482.67	0.00	366	6.71	366	6.71	0.56	343	0.00	6.81
	25	1062	3600.00	28.61	1757	65.44	1066.5	0.42	1.04	1062	0.00	13.81
$S_{_21}$	8	464	0.09	0.00	464	0.00	464	0.00	0.07	464	0.00	0.61
	10	609	0.45	0.00	615	0.99	609	0.00	0.11	609	0.00	1.05
	15	1243	463.31	0.00	1248	0.40	1243	0.00	0.22	1243	0.00	2.84
	20	2929	3600.00	67.15	3161	7.92	2929	0.00	0.55	2929	0.00	7.11
	25	5057	3600.00	96.03	5036	2.71	4909	0.12	0.62	4903	0.00	11.17
$S_{_22}$	8	189	0.11	0.00	194	2.65	189.5	0.26	0.06	189	0.00	0.59
	10	569	1.93	0.00	569	0.00	569	0.00	0.11	569	0.00	1.06
	15	540	120.70	0.00	540	0.00	540	0.00	0.25	540	0.00	3.03
	20	1352	3600.00	21.45	1352	0.00	1352	0.00	0.49	1352	0.00	5.95
	25	191	3600.00	11.52	302	58.12	203.6	6.60	0.87	191	0.00	9.83
$S_{_31}$	8	664	0.09	0.00	672	1.20	664	0.00	0.07	664	0.00	0.73
	10	856	1.28	0.00	858	0.23	856	0.00	0.13	856	0.00	1.22
	15	2316	833.79	0.00	2437	5.22	2323.1	0.31	0.37	2316	0.00	3.56
	20	2332	3600.00	44.51	2547	9.31	2331.4	0.06	0.73	2330	0.00	8.09
	25	2354	3600.00	80.46	2503	6.33	2360.5	0.28	0.98	2354	0.00	12.15
$S_{_32}$	8	319	0.06	0.00	319	0.00	319	0.00	0.07	319	0.00	0.68
	10	217	0.45	0.00	236	8.76	232.2	7.00	0.11	217	0.00	1.17
	15	50	23.65	0.00	80	60.00	62.4	24.80	0.41	50	0.00	3.73
	20	378	1106.25	0.00	500	32.28	378.5	0.13	0.79	378	0.00	5.92
	25	218	3600.00	34.40	234	24.47	218.2	16.06	0.86	188	0.00	11.16
Average			1357.14	17.48		11.63		2.32	0.42		0.00	4.93

Table 2: Comparison of the SWSP, the VNS and the GVNS for large-sized instances.

Groups	n	SWSP		VNS				GVNS			
		Objective Value	RPD(%)	Mean	RPD(%)	MAD(%)	Time(s)	Mean	RPD(%)	MAD(%)	Time(s)
$S_{_11}$	50	20820	15.90	18200	1.31	0.58	5.67	17971	0.04	0.06	120.13
	60	23895	15.24	20995	1.25	0.89	13.17	20761	0.13	0.15	224.42
	70	42027	19.00	36339	2.90	0.67	14.86	35324	0.02	0.01	338.22
	80	67739	24.61	55266	1.66	0.42	20.30	54400	0.07	0.09	533.01
	90	82112	16.35	72307	2.46	0.44	25.02	70675	0.14	0.04	634.76
$S_{_12}$	100	78452	17.05	67609	0.87	0.26	58.67	67215	0.28	0.19	931.67
	50	8501	29.27	6817.3	3.67	2.18	5.63	6603.1	0.41	0.74	114.08
	60	1208	4546.15	437.3	1581.92	29.66	13.76	29.6	13.85	17.03	145.59
	70	4047	289.13	2279.4	119.17	8.48	16.69	1065.4	2.44	2.22	274.37
	80	9808	56.20	7365.1	17.30	6.44	20.89	6318.6	0.63	0.98	440.14
$S_{_21}$	90	8731	110.13	5015.7	20.71	5.89	37.52	4212	1.37	1.08	625.96
	100	4961	158.92	3211.2	67.60	9.51	48.56	1966.8	2.65	3.05	851.83
	50	20980	6.28	19788	0.24	0.18	5.19	19740	0.00	0.00	69.45
	60	29281	5.93	27694	0.19	0.09	8.19	27642	0.00	0.00	116.10
	70	46270	8.77	42636	0.23	0.15	14.47	42554	0.04	0.05	209.57
$S_{_22}$	80	50660	10.42	46103	0.49	0.23	15.67	45880	0.00	0.00	225.04
	90	57859	13.55	51118	0.32	0.11	29.56	50958	0.00	0.01	406.09
	100	80275	11.98	71975	0.40	0.18	30.83	71695	0.01	0.01	583.66
	50	320	57.64	212.1	4.48	5.15	4.77	203	0.00	0.00	43.82
	60	1647	55.23	1198.3	12.94	9.48	7.88	1061	0.00	0.00	114.46
$S_{_31}$	70	5063	8.91	4797.4	3.19	1.55	11.64	4649	0.00	0.00	152.83
	80	2535	34.98	2161.5	15.10	3.05	17.96	1878	0.00	0.00	219.78
	90	11089	17.18	9785.6	3.41	1.64	33.16	9463.1	0.00	0.00	444.46
	100	3467	23.34	3007.8	7.00	1.45	24.15	2811	0.00	0.00	363.21
	50	15639	14.35	13817	1.03	0.35	7.56	13676	0.00	0.00	91.41
$S_{_32}$	60	31578	9.49	28930	0.31	0.06	7.76	28842	0.00	0.00	143.20
	70	38983	19.39	32953	0.92	0.33	19.41	32660	0.02	0.03	290.32
	80	49429	27.75	39330	1.65	0.41	27.62	38725	0.08	0.06	398.79
	90	75215	17.39	64855	1.22	0.63	32.61	64123	0.08	0.06	623.88
	100	76221	21.44	63412	1.03	0.28	30.48	62765	0.00	0.00	641.97
$S_{_32}$	50	1175	48.92	1121	42.08	3.07	5.51	789	0.00	0.00	64.15
	60	999	239.80	688.7	134.25	8.09	8.67	301.2	2.45	0.96	113.51
	70	3867	94.03	2914.6	46.24	7.29	15.94	1994	0.05	0.05	244.40
	80	22127	21.74	18782	3.34	1.35	22.28	18179	0.02	0.02	517.98
	90	7729	45.34	5791.5	8.90	2.17	31.23	5408	1.69	1.20	647.39
Average	100	5061	162.64	4207.2	118.33	7.05	34.63	1955.8	1.49	1.18	540.83
			174.29		61.89	3.33	20.22		0.78	0.81	347.23